

# 3D渲染插件MetaKit 集成Doc —iOS ver2.2

## 准备环境

在开始集成MetaKit前，请确保开发环境满足以下要求：

- Xcode 13.0 或以上版本。
- iOS 14.0及以上的系统版本，且支持音视频功能的iOS真机设备。
- 设备的前置摄像头与麦克风功能正常。
- Apple 开发者账号。
- 安装 Cocoapods。

## 功能列表

- 虚拟人avatar
  - avatar切换
  - avatar面捕驱动
  - avatar捏脸
  - avatar换装
  - avatar视角切换
- 遮脸头套animoji
  - animoji切换
  - animoji面捕驱动
- 挂件贴纸sticker
  - 人脸装饰：眼镜、面纱、口罩、帽子
- 背景特效
  - 2d图片背景替换
  - 360全景图背景替换
  - 3d场景背景替换
- 灯光/律动特效
  - 打光特效：3D打光、氛围灯、广告灯
  - 人像边缘特效：火焰律动、光线律动
  - 其它：波纹、极光

Agora提供美术资源的设计与生产规范，需要自定义美术资源可联系技术支持。

## 集成SDK

### 1 API-Examples

请跑通Agora的RTC demo: API-Examples，然后基于此定制meta需求，参照<https://doc.shengwang.cn/doc/rtc/ios/get-started/run-demo>

### 2 集成AgoraSDK 4.3.2.3版本

可选择cdn直接下载，或使用cocoapods依赖

【cdn下载】

[https://download.agora.io/sdk/release/Agora\\_Native\\_SDK\\_for\\_iOS\\_rel.v4.3.2.3\\_44713\\_FULL\\_20240808\\_1505\\_326360.zip](https://download.agora.io/sdk/release/Agora_Native_SDK_for_iOS_rel.v4.3.2.3_44713_FULL_20240808_1505_326360.zip)

【cocoapods依赖】

```
pod 'AgoraRtcEngine_Special_iOS', '4.3.2.3'
```

### 3 集成Agora插件

full包里已包含面捕插件与分割插件，metakit插件请在资源目录SDK里获取。

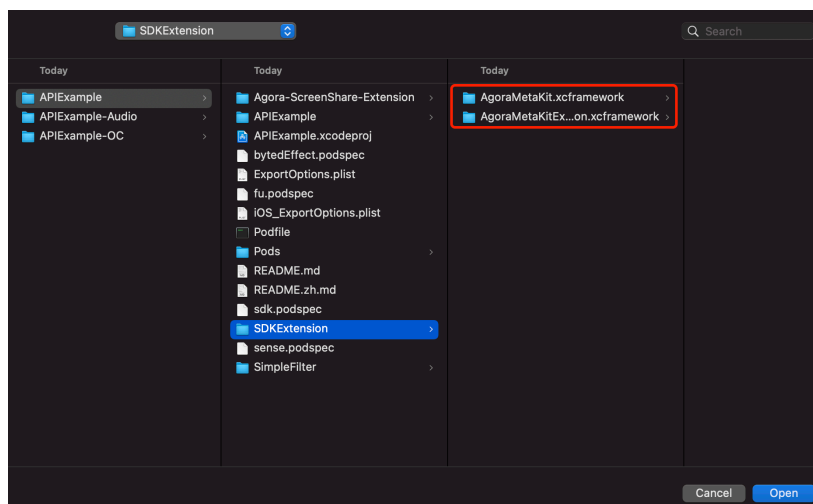
**注意：**

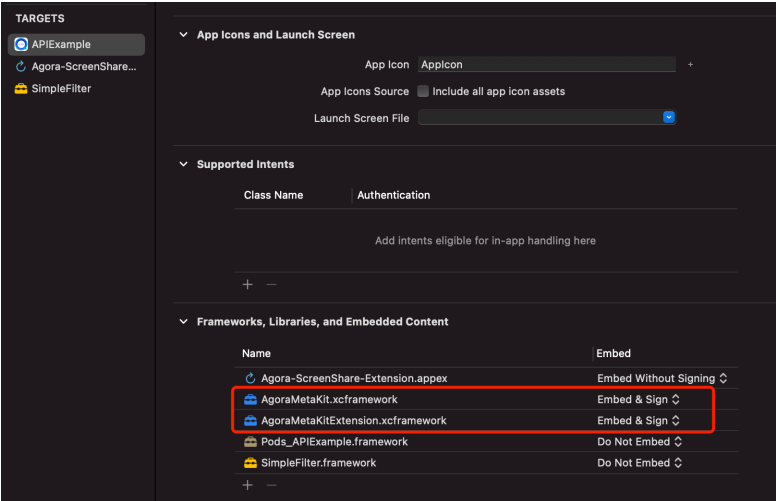
Agora提供XCFramework形式的SDK，内含“arm64”的架构，不包含“armv7”的架构

不同功能需要集成的插件不太相同，请以下表为准：

| 功能                         | 插件  |
|----------------------------|---|
| 虚拟人avatar                  | metakit插件：<br><br>AgoraMetaKitExtension.xcframework<br><br>AgoraMetaKit.xcframework<br><br>面捕插件：<br><br>AgoraFaceCaptureExtension.xcframework       |
| 遮脸头套animoji<br>挂件贴纸sticker | metakit插件：<br><br>AgoraMetaKitExtension.xcframework<br><br>AgoraMetaKit.xcframework<br><br>面捕插件：<br><br>AgoraFaceCaptureExtension.xcframework       |
| 背景特效                       | metakit插件：<br><br>AgoraMetaKitExtension.xcframework<br><br>AgoraMetaKit.xcframework<br><br>分割插件：<br><br>AgoraVideoSegmentationExtension.xcframework |
| 律动特效                       | metakit插件：<br><br>AgoraMetaKitExtension.xcframework<br><br>AgoraMetaKit.xcframework<br><br>分割插件：<br><br>AgoraVideoSegmentationExtension.xcframework |

在APIExample.xcproject下创建一个“SDKExtension”目录把插件拷贝进去，然后在 General-Frameworks, Libraries, and Embedded Content上添加依赖





## 资源获取

资源在包里的assets目录下

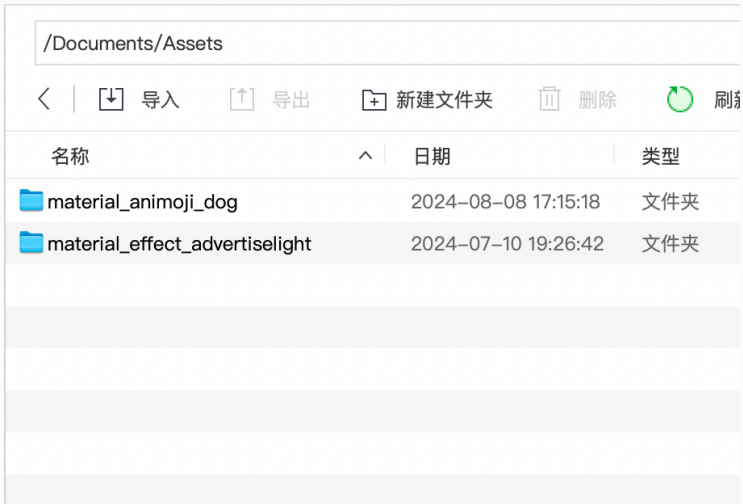
| 资源名称                   | 说明   | 大小   |
|------------------------|--|--|
| material_avatar_xxx    | 虚拟人Avatar的分包资源。包含虚拟人形像"girl", "boy", "huamulan"。可使用面捕驱动，捏脸，换装等能力，集成时需要将该资源的绝对路径设置给MetaKit。 | <ul style="list-style-type: none"><li>虚拟人"girl"（支持捏脸换装）</li><li>虚拟人"boy"（仅支持捏脸）</li><li>虚拟人"huamulan"（不支持捏脸换装）</li></ul>                                   |
| material_package_xxx   | 虚拟人Avatar的套装资源，包含虚拟人形像"girl"的3套衣服和一套捏脸数据。  | <ul style="list-style-type: none"><li>虚拟人"girl"套装1</li><li>虚拟人"girl"套装2</li><li>虚拟人"girl"套装3</li></ul>   |
| material_animoji_xxx   | 遮脸头套Animoji的分包资源。包含头套"dog", "girlhead"。可用面捕驱动能力，集成时需要将该资源的绝对路径设置给MetaKit。                  | <ul style="list-style-type: none"><li>头套"dog"</li><li>头套"girlhead"</li><li>头套"arkit"</li></ul>   |
| material_sticker_xxx   | 挂件贴纸Sticker的分包资源。  | <ul style="list-style-type: none"><li>挂件"眼镜"</li><li>挂件"口罩"</li><li>挂件"面纱"</li><li>挂件"龙头帽"</li></ul>   |
| material_bg_effect_xxx | 背景特效的分包资源。   | <ul style="list-style-type: none"><li>"全景图"背景</li><li>"3D房间"背景</li></ul>   |
| material_effect_xxx    | 特效Effect的分包资源。包含“3D打光“，”氛围灯“，”广告灯“，”火焰律动“，”光线律动“，”极光“，”屏幕波纹“，集成时需要将该资源的绝对路径设置给MetaKit      | <ul style="list-style-type: none"><li>特效"3D打光"</li><li>特效"氛围灯"</li><li>特效"广告灯"</li><li>特效"火焰律动"</li><li>特效"光线律动"</li><li>特效"极光"</li><li>特效"屏幕波纹"</li></ul> |

|              |               |  |
|--------------|---------------|--|
| 捏脸换装部位信息.txt | Avatar捏脸换装的信息 |  |
| 素材.md        | 素材包的对照目录      |  |

**注意：**

资源的使用是不同的素材分包资源合并使用的，客户根据自己想要的素材功能把目录平行放在一个根目录。

eg：想test 狗狗头套与广告灯特效的能力，把"material\_animoji\_dog"和"material\_effect\_advertiselight"，放到iOS沙盒的同一级根目录下。



## 快速开始

- 1 创建一个ViewController.swift用于处理Meta的逻辑。
- 2 创建AgoraRtcEngine，并监听其事件回调。

```

let cfg = AgoraRtcEngineConfig()
cfg.appId = ...
cfg.eventDelegate = self
agoraKit = AgoraRtcEngineKit.sharedEngine(with: cfg, delegate: self)

extension XXXController: AgoraMediaFilterEventDelegate {
    func onEvent(_ provider: String?, extension: String?, key: String?,
value: String?) {
        //
    }

    func onExtensionStarted(_ provider: String?, extension: String?) {
        //
    }

    func onExtensionStopped(_ provider: String?, extension: String?) {
        //
    }

    func onExtensionError(_ provider: String?, extension: String?,
error: Int32, message: String?) {
        //
    }
}

```

### 3 注册并插件并设置好参数，需要保证依赖插件的注册在metakit插件之前

面捕插件+metakit:

```
//
agoraKit?.registerExtension(withVendor:
"agora_video_filters_face_capture", extension: "face_capture",
sourceType: AgoraMediaSourceType.primaryCamera)
agoraKit?.registerExtension(withVendor: "agora_video_filters_metakit",
extension: "metakit", sourceType: AgoraMediaSourceType.primaryCamera)

// (license)
agoraKit?.enableExtension(withVendor:
"agora_video_filters_face_capture", extension: "face_capture", enabled:
true)
agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_face_capture",
extension:
"face_capture",
key:
"authentication_information",
value: "{ \"
company_id\": \"xxx\", \"license\": \"xxxx\" }",
sourceType:
AgoraMediaSourceType.primaryCamera)
// metakit
agoraKit?.enableExtension(withVendor: "agora_video_filters_metakit",
extension: "metakit", enabled: true)
agoraKit?.setExtensionPropertyWithVendor("agora_video_filters_metakit",
extension: "metakit", key: "initialize", value: "{}")
```

面捕license试用:

company\_id: agoraDemo

license: HIYW7P5x+IE8H1lNlEKfDAWOYc+rFIyhQ/QEDomcQhBoQrlHpMWqL9+HGAOBjBWZtcPHjol4N  
sRDlUo6UQ85ib/XSH+1MFlpZT/r5nCTADTkOclPqfMKic4gNjr4sb3PD3v4EaZ89tZoYMw0LM6rd  
jtlySH+8yaWKS+hKKHY=

分割插件+metakit:

```
//
agoraKit.setParameters("{\"rtc.video(seg_before_exts\":true}")
//
agoraKit?.registerExtension(withVendor: "agora_video_filters_metakit",
extension: "metakit", sourceType: AgoraMediaSourceType.primaryCamera)

//
let bg_src = AgoraVirtualBackgroundSource()
bg_src.backgroundSourceType = .none
let seg_prop = AgoraSegmentationProperty()
seg_prop.modelType = .agoraAi
agoraKit?.enableVirtualBackground(true, backData: bg_src, segData:
seg_prop)

// metakit
agoraKit?.enableExtension(withVendor: "agora_video_filters_metakit",
extension: "metakit", enabled: true)
agoraKit?.setExtensionPropertyWithVendor("agora_video_filters_metakit",
extension: "metakit", key:"initialize", value:"{")
```

#### 4 创建一个SceneView，用于展示Meta的内容

```
let metakit: MetaKitEngine = MetaKitEngine.sharedInstance()
var frame = CGRect(x: 0, y: 0, width: 360, height: 640);
// sceneViewframe(9:16)
//
let sceneView = metakit.createSceneView(frame)
// viewios UIViewUIaddview
// agorartc5-8
// view
let address = unsafeBitCast(sceneView!, to: Int64.self)
let value = enable ? 1 : 0
agoraKit?.setExtensionPropertyWithVendor(vender, extension:
extension_name,
key:"enableSceneVideo",
value:"{\"view\": \"\"(address)

\", \"enable\": \"\"(value)\"")
```

#### 5 加载资源场景，在执行具体时按功能获取对应的资源。

注：加载资源可以反复执行切换，不需要load与unload配对：loadMaterial(pathA)->loadMaterial(pathB)->loadMaterial(pathC) → ... → unloadMaterial()

```

//
// pathunity
let address = unsafeBitCast/avatarView!, to: Int64.self)
let dict = ["path": path, "view": String(address)]
let data = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let dictInfo = String(data: data!, encoding: String.Encoding.utf8)

self.agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_metakit", extension: "metakit",
                                key:"loadMaterial",
                                value:dictInfo ?? "")

// 6-7
let dict = ["path": path]
let data = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let dictInfo = String(data: data!, encoding: String.Encoding.utf8)

self.agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_metakit", extension: "metakit",
                                key:"unloadMaterial",
                                value:dictInfo ?? "")

```

**6 移除一个SceneView (注, 如果没有动态删除view的需求, 销毁引擎 (步骤8) 会自动remove 所有的 sceneview)**

```

let address = unsafeBitCast(view, to: Int64.self)
agoraKit?.setExtensionPropertyWithVendor("agora_video_filters_metakit",
extension: "metakit",
                                key:"removeSceneView",
                                value:"{\\"view\\":\\"(address)
\\"}")

```

## 7 销毁渲染引擎

```

// view
sceneView?.removeFromSuperview()
metakit.removeSceneView(sceneView?)
//
agoraKit?.setExtensionPropertyWithVendor("agora_video_filters_metakit",
extension: "metakit",
                                key:"destroy", value:"{ }")

```

**8 关于引擎状态的同步, 在1中的代理函数, onEvent会透传引擎状态, onExtensionError会透传错误信息**

```

func onEvent(_ provider: String?, extension: String?, key: String?,
value: String?) {
    if(provider == "agora_video_filters_metakit" && extension ==
"metakit") {
        guard let status = key else {
            return
        }
        switch status {
            case "initializeFinish":    //
            case "unityLoadFinish":    //
            case "materialLoaded":      // response
            case "materialUnLoaded":    // response, 7
                DispatchQueue.main.async {
                    // maindispatch
                }
            break;
            default:
        }
    }
}

func onExtensionError(_ provider: String?, extension: String?, error:
Int32, message: String?) {
    if(provider == "agora_video_filters_metakit" && extension ==
"metakit") {
        print("[MetaKit] onExtensionError, Code: \(error), Message: \(
message ?? "")");
    }
}

```

## 虚拟人avatar

- 1 获取avatar资源(material\_avatar\_xx)，导入iOS沙盒，按“快速开始”中的步骤在loadMaterial时设置资源的path。
- 2 增加一个SceneView，并在上面增加Avatar的内容，按如下代码：

```

// avatar
let address = unsafeBitCast/avatarView!, to: Int64.self)
let dict = ["path": path_to_avatar, "view": String(address)]
let data = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let dictInfo = String(data: data!, encoding: String.Encoding.utf8)

self.agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_metakit", extension: "metakit",
                                key:"loadMaterial",
                                value:dictInfo ?? "")

// avatar
// viewmode 0- 1- 2-
agoraKit?.setExtensionPropertyWithVendor("agora_video_filters_metakit",
extension: "metakit",
                                key:"setCamera",
                                value:"{\\"viewMode\\":\\"\\
(viewmode)\\"}")
// avatar
// avatar material_package_dress_xx,
let dict = ["path": path_to_dress, "view": String(address)]
let data = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let dictInfo = String(data: data!, encoding: String.Encoding.utf8)

self.agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_metakit", extension: "metakit",
                                key:"loadMaterial",
                                value:dictInfo ?? "")

// avatar
// avatar material_package_faceshape_xx,
let dict = ["path": path_to_faceshape, "view": String(address)]
let data = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let dictInfo = String(data: data!, encoding: String.Encoding.utf8)

self.agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_metakit", extension: "metakit",
                                key:"loadMaterial",
                                value:dictInfo ?? "")

```

### 3 avatar捏脸换装的部位参数见：

assets/捏脸换装部位信息.txt

eg1: 文档里shapes信息为可捏脸的部位名，捏脸程度为百分比（0-100）

```

"avatar": "girl",
"blendshape": [
  {
    "type": "Face",
    "shapes": [
      {
        "key": "FE_raise_1",
        "ch": "额头的突出"
      },
      {
        "key": "FE_raise_2",
        "ch": "额头的凹陷"
      },
      {
        "key": "TP_raise_1",
        "ch": "太阳穴的突出"
      },
      {
        "key": "TP_raise_2",
        "ch": "太阳穴的凹陷"
      },
      {
        "key": "CK_raise_1",
        "ch": "脸颊的突出"
      },
      {
        "key": "CK_raise_2",

```

eg2: 文档里assets下为换装的部位

```

"avatar": "girl",
"resources": [
  {
    "id": 100,
    "name": "头发",
    "assets": [
      10000,
      10001,
      10002
    ]
  },
  {
    "id": 101,
    "name": "眉毛",
    "assets": [
      10100,
      10101,
      10102
    ]
  },

```

## 遮脸头套animoji / 挂件Sticker

1 获取animoji资源(material\_animoji\_xx)资源 或 sticker资源(material\_sticker\_xx), 导入iOS沙盒, 按“快速开始”中的步骤在loadMaterial时设置资源的path。

2 增加一个SceneView, 并在上面增加Animoji、Sticker的内容, 按如下代码:

```
//
let address = unsafeBitCast/avatarView!, to: Int64.self)
let dict = ["path": path, "view": String(address)]
let data = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let dictInfo = String(data: data!, encoding: String.Encoding.utf8)

self.agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_metakit", extension: "metakit",
                                key:"loadMaterial",
                                value:dictInfo ?? "")

// animojiquality 0- 1- 2-
let dict = ["general": quality]
let value = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let str = String(data: value!, encoding: String.Encoding.utf8) ?? ""
self.agoraKit?.setExtensionPropertyWithVendor("agora_video_filters_metakit",
extension: "metakit",
                                key: "setRenderQuality",
                                value: str)
```

## 背景特效

1 获取背景特效资源(material\_bgeffect\_xx)，导入iOS沙盒，按“快速开始”中的步骤在loadMaterial时设置资源的path。

2 增加一个SceneView，并在上面增加背景特效的内容，按如下代码：

```
//
let address = unsafeBitCast/avatarView!, to: Int64.self)
let dict = ["path": path, "view": String(address)]
let data = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let dictInfo = String(data: data!, encoding: String.Encoding.utf8)

self.agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_metakit", extension: "metakit",
                                key:"loadMaterial",
                                value:dictInfo ?? "")
```

## 灯光/律动特效

1 获取背景特效资源(material\_effect\_xx)，导入iOS沙盒，按“快速开始”中的步骤在loadMaterial时设置资源的path。

2 增加一个SceneView，并在上面增加灯光/律动特效的内容，按如下代码：

```
//
let address = unsafeBitCast/avatarView!, to: Int64.self)
let dict = ["path": path, "view": String(address)]
let data = try? JSONSerialization.data(withJSONObject: dict, options:
[])
let dictInfo = String(data: data!, encoding: String.Encoding.utf8)

self.agoraKit?.setExtensionPropertyWithVendor
("agora_video_filters_metakit", extension: "metakit",
                                key:"loadMaterial",
                                value:dictInfo ?? "")

// 1001"3D"
let light_dict = ["id": 1001, "param": ["intensity": 2.0, "scale": 0.3]
as [String : Any], "enable": true] as [String : Any]
let light_data = try? JSONSerialization.data(withJSONObject:
light_dict, options: [])
let light_info = String(data: light_data!, encoding: String.Encoding.
utf8)
agoraKit?.setExtensionPropertyWithVendor("agora_video_filters_metakit",
extension: "metakit",
                                key:"setEffectVideo",
                                value:light_info!)
```

### 3 特效参数:

参数名称解释:

| 参数         | 值类型    | 解释      |
|------------|--------|---------|
| color      | int64  | 普通颜色    |
| startColor | int64  | 渐变颜色-起始 |
| endColor   | int64  | 渐变颜色-结束 |
| intensity  | float  | 强度      |
| scale      | float  | 缩放因子    |
| speed      | float  | 速度      |
| size       | float  | 尺寸      |
| range      | float  | 场景扫描范围  |
| text       | string | 文本内容    |
| animation  | int    | 动画类型    |

| setEffectVideo |       |                  |  |                                       |                                 |                        |
|----------------|-------|------------------|--|---------------------------------------|---------------------------------|------------------------|
| id             | name  | str              | param  | type                                  | 建议值                             | 默认值                    |
| 1001           | 3D打光  | 3dLight          | color # intensity # scale                        | int64 # float # float                 | \ # 1.0~2.0 # 0.3~0.6           | \ # 1.6 # 0.4          |
| 1002           | 屏幕波纹  | screenRipple     | color # speed # scale                            | int64 # float # float                 | \ # -0.2~0.2 # 3.0~6.0          | \ # -0.12 # 4.0        |
| 1003           | 极光    | aurora           | color # intensity                                | int64 # float                         | \ # 0.8~1.5                     | \ # 1.0                |
| 2001           | 紫色火焰  | purpleFlame      | color # intensity                                | int64 # float                         | \ # 0.2~1.5                     | \ # 0.2                |
| 3001           | 氛围灯光组 | lightGroup       | quality  | int                                   | 0/1/2                           | 0                      |
| 3002           | 广告灯   | advertisinglight | startColor # endColor # size # intensity # range | int64 # int64 # float # float # float | \ # \ # 8~15 # 100~1000 # 10-40 | \ # \ # 10 # 1000 # 15 |

## 注意事项

1. metakit库跨SDK大版本（4.2.x, 4.3.x, 4.4.x）是无法保证兼容的，请依据自己集成的SDK版本获取对应的library，以免出现稳定性问题。
2. 资源如果是走下载解压的方式，请业务层开发自己保证下载解压后资源的完备性。如果目录不对，或有资源bundle丢失，加载时会出现异常。
3. 收到引擎状态的消息不是主线程，如果在这里操作，请dispatch\_async到主线程执行操作，避免线程问题。
4. 请确保在收到确切的引擎状态后，执行对应的操作，以免出现不必要的时序问题：
  1. 在收到unityLoadFinish后表示引擎启动完成，可以触发资源加载loadMaterial。
  2. 在收到materialLoaded后进行一些状态切换（如特效参数设置），行为变更（如捏脸换装）。
  3. 在收到materialUnLoaded后仅可执行引擎销毁，不可重复进行loadMaterial操作。
  4. 如果页面有进行metakit initialize，在退出页面前确保有发送destroy，保证引擎的正常生命周期，减少性能和内存的影响。